The Unknown Coder: The AI Code Risk

Tyson Shannon

CIS, Minnesota State University-Mankato

CIS202W: Computers in Society

April 10, 2026

The Unknown Coder: The AI Code Risk

With the rise of AI, its use for generation production level code has only increased, bringing with it numerous new risks. The term “script kiddie” rose among the hacker community in the early 1990’s to define young hackers who use pre-made exploits to try and gain access to IT infrastructure without fully understanding how the code they use works (Mistry, 2024). For decades “scrip kiddie” has been thrown around as an insult, and many viewed the practice as relatively harmless due to the unsophisticated nature of the attacks. However, with the rise of AI, this hacker term has taken a new and more concerning meaning, as many programmers and companies swarm to LLM’s to replace large development teams. In the modern day “script kiddies” have shifted from attempting to exploit systems to creating vulnerable programs with the same level of technical ignorance but an increased sophistication due to AI assistance. As a result, AI generated code represents the greatest modern cyber security threat because it introduces unknown vulnerabilities at an increasing scale, reduces developer accountability, and speeds up insecure development without the proper safeguards usually employed.

A recent report made by a Massachusetts based application security company discovered that 45% of all AI generated coding tasks introduced security flaws into the code (Veracode, 2025). The report also found that despite larger more modern models improving in writing proper syntax, models remained consistent in their poor security performances. One of the reasons for this is that AI is trained on code repositories across the internet, many of which are syntactically correct but are outdated or intentionally vulnerable for educational purposes. This can be seen in a language like Java which predates the recognition of SQL injection as a vulnerability, and therefore Java tasks contain this security vulnerability more often when generated by AI (Veracode, 2025). AI also lacks the deep context and knowledge of a program's properties to properly secure the systems it designs. An academic report from China found that while over 75% of the vulnerabilities in code in the wild were introduced by humans, AI introduced a higher proportion of vulnerabilities in C# and PHP suggesting it is more prone to make errors in those languages (Wang et al., 2025). Humans have far more programmers of varying abilities and with far more responsibilities which could explain the high vulnerability introduction rates. This means as more developer responsibilities are replaced with the few AI models on the market, we could see these results quickly shift. This report also noticed AI introducing certain vulnerabilities more than others, especially when dealing with input sanitization similar to what was discussed above with the SQL injection errors. It was also found that the risk scores of AI introduced vulnerabilities mimicked that of humans suggesting a relation due to training data. This means that while the security flaws created are not technically any worse than those by humans, it also means that AI code will likely never be safer than human code.

Outside of code vulnerabilities, another security issue that arises is the lack of accountability and transparency for the design decisions made. As we automate our responsibilities, mistakes are no longer assigned to individuals or teams but to technology and can therefore be written off as a random and unavoidable technical glitch. However, these mistakes are avoidable with proper oversight and therefore accountability needs to remain with the developers generating the code and the corporate structures that are supposed to oversee their actions. A faulty tool does not legally exempt someone when individuals are harmed by their product, and system architects should continue to be held responsible for data leaks, service outages, and cyber theft. The National Institute of Standards and Technology has laid out a lifecycle for proper use of AI systems that involves verifying and validating systems before deployment and continued monitoring of systems after deployment (NIST, 2023). The AI system lifecycle melds well with a traditional software development lifecycle and requires that developers maintain an understanding of their programs and therefore remain accountable. The NIST also states that for an AI system to be considered trustworthy it must be “valid and reliable, safe, secure and resilient, accountable and transparent, explainable and interpretable, privacy-enhanced, and fair with harmful bias managed” (NIST, 2023). In most cases, the use of AI generated code does not meet these requirements and therefore cannot be viewed as trustworthy. AI code without human intervention and oversight fails to meet security standards and puts developers and companies in legal and ethical hot water when security incidents occur.

Some, however, disagree with these risks. In a Canadian academic article on the GitHub Copilot code generator and how it pairs with developers, researchers acknowledge the model's buggy nature but conclude it still improves efficiency in advanced developers (Dakhel et al., 2023). One reason for this conclusion that they give is their experiment that demonstrated Copilot’s buggy code was easier to repair compared to the buggy code made by students, however the analysis was done with basic algorithms and inexperienced human developers. As codebases grow and become more unique, so does their complexity. If the complex code is developed by humans, then their long-term memory and ability to grasp context allows for easier debugging, whereas AI code becomes harder to follow, and models often become confused. It is also expected for AI to outperform new student developers with limited experience, especially if companies are planning to replace programmers with these models. Therefore, I would argue when competing against practiced coders on a complex system, debugging AI code lowers efficiency because it is less understood and less capable of explaining itself. As a result, AI code would be more likely to have bugs slip through the debugging process. The article also believes quality of the code is related to quality of the prompt, insinuating that AI code would be less buggy with a better human input. However, as we have discussed, the quality of AI code is directly related to the human code it's trained on and therefore there is no decrease in risk depending on the ability and experience of the developer using the tool. The article concludes that these models are only a liability if novice developers rely on them because of their inability to properly assess the code, but AI code is often deployed by professionals without a complete understanding of how it works creating the same level of risk.

A potential solution to these problems is a code generation framework referred to as GSecGen that has been designed to ensure secure generation, analysis, and debugging of AI code (Karuppan et al., 2025). This framework prioritizes security in each step of the code generation process and will regenerate code until vulnerabilities are patched, automating assessment and securing code from the point of creation. This framework has been shown to improve functional correctness by up to 9.1% and reduce vulnerability creation by up to 14.7%. If we combine these new innovative systems with the NIST guidelines discussed prior and with experienced human moderation and intervention, code creation could become far more secure than ever before. What many have failed to realize is that the solution to creating better, more secure code is the combination of new and existing practices opposed to the replacement of one for another. Both AI and human developers are capable of failure and prone to error, but when properly combined, their vulnerability assessments can be extremely successful in preventing the deployment of unsecure code. It will remain essential, however, that developers remain aware and accountable for their code and consistently test and monitor their systems for error.

In conclusion AI generated code is the most dangerous cyber security threat of today because of its increasing popularity, its tendency to introduce vulnerabilities, its use to offload responsibility and understanding of a codebase from a human, and its use to speed through development without following proper standards and practices. In practice, AI is more likely to introduce security flaws, including simple ones, due to the vulnerable code it’s trained on. Since AI is trained on human code, it will never be more secure to use an AI developer over a human one. The only way to avoid potentially dangerous bugs in our systems is to fully understand the code we create and continually run vulnerability assessments while following development best practices. If we combine security focused AI with human ability, security will drastically improve across the industry.

References

Mistry, S. (2024, September 25). Script Kiddies: A Review. Packetlabs. https://www.packetlabs.net/posts/emergence-of-script-kiddies/

Veracode. (2025). GenAI Code Security Report: Assessing the Security of Using LLMs for Coding. https://www.veracode.com/wp-content/uploads/2025_GenAI_Code_Security_Report_Final.pdf

Wang, B., Yu, W., Zhong, Y., Yu, H., Lian, K., Lu, C., Zheng, H., Zhang, D., & Li, H. (2025). AI Code in the Wild: Measuring Security Risks and Ecosystem Shifts of AI-Generated Code in Modern Software (arXiv:2512.18567). ArXiv. https://doi.org/10.48550/arXiv.2512.18567

National Institute of Standards and Technology. (2023, January). Artificial Intelligence Risk Management Framework (AI RMF 1.0) (NIST AI 100-1). https://doi.org/10.6028/NIST.AI.100-1

Dakhel, A. M., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., & Jiang, Z. M. (2023, September). GitHub Copilot AI Pair Programmer: Asset or Liability? Journal of Systems and Software, 203, 111734. https://doi.org/10.1016/j.jss.2023.111734

Karuppan, S., Shanmugasundaram, S., Mouriya, J., & Sanjeeth, J. (2025). Generalizable secure code generation framework for robust software development practices. In Proceedings of the 5th International Conference on Mobile Networks and Wireless Communications (ICMNWC). https://doi-org.ezproxy.mnsu.edu/10.1109/ICMNWC66779.2025.11354349